FORM PTO-1390 U.S. DEPARTMENT OF COMMERCE PATENT AND TRADEMARK OFFICE		ATTORNEY'S DOCKET NUMBER
TRANSMITTAL LETTER TO THE UNITED STATES DESIGNATED/ELECTED OFFICE (DO/EO/US) CONCERNING A FILING UNDER 35 U.S.C. 371		I01/3 U. S. APPLICATION NO. (If known, see 37 CFR 1.5) 09/937315
INTERNATIONAL APPLICATION NO. PCT/IL00/00178	INTERNATIONAL FILING DATE March 20, 2000	PRIORITY DATE CLAIMED March 22, 1999
TITLE OF INVENTION AUTOMATIC INTERFACE GENERATION FOR AN ENTERPRISE RESOURCE		
APPLICANT(S) FOR DO/EO/US ILAN HADAS		

Applicant herewith submits to the United States Designated/Elected Office (DO/EO/US) the following items and other information:

1. ☒ This is a **FIRST** submission of items concerning a filing under 35 U.S.C. 371.
2. ☐ This is a **SECOND** or **SUBSEQUENT** submission of items concerning a filing under 35 U.S.C. 371.
3. ☐ This express request to being national examination procedures (35 U.S.C. 371(f)) at any time rather than delay examination until the expiration of the applicable time limit set in 35 U.S.C. 371(b) and PCT Articles 22 and 39(1).
4. ☒ A proper Demand for International Preliminary Examination was made by the 19th month from the earliest claimed priority date.
5. ☒ A copy of the International Application as filed (35 U.S.C. 371(c)(2))
   a. ☒ is transmitted herewith (required only if not transmitted by the International Bureau).
   b. ☐ has been transmitted by the International Bureau.
   c. ☐ is not required, as the application was filed in the United States Receiving Office (RO/US).
6. ☐ A translation of the International Application into English (35 U.S.C. 371(c)(2)).
7. ☒ Amendments to the claims of the International Application under PCT Article 19 (35 U.S.C. 371(c)(3))
   a. ☒ are transmitted herewith (required only if not transmitted by the International Bureau).
   b. ☐ have been transmitted by the International Bureau.
   c. ☐ have not been made; however, the time limit for making such amendments has NOT expired.
   d. ☐ have not been made and will not be made.
8. ☐ A translation of the amendments to the claims under PCT Article 19 (35 U.S.C. 371(c)(3)).
9. ☐ An oath or declaration of the inventor(s) (35 U.S.C. 371(c)(4)).
10. ☐ A translation of the annexes to the International Preliminary Examination Report under PCT Article 36 (35 U.S.C. 371(c)(5)).

Items 11. to 16. below concern document(s) or information included:

11. ☒ An Information Disclosure Statement under 37 CFR 1.97 and 1.98.
12. ☐ An assignment document for recording. A separate cover sheet in compliance with 37 CFR 3.28 and 3.31 is included.
13. ☐ A FIRST preliminary amendment.
    ☐ A SECOND or SUBSEQUENT preliminary amendment.
14. ☐ A substitute specification.
15. ☐ A change of power of attorney and/or address letter.
16. ☐ Other items or information:

U. S. APPLICATION NO. (if known, see 37 CFR 1.5)	INTERNATIONAL APPLICATION NO.	ATTORNEY'S DOCKET NUMBER
09/937315	PCT/IL00/00178	I01/1

17. ☐ The following fees are submitted:	CALCULATIONS PTO USE ONLY	
BASIC NATIONAL FEE (37 CFR 1.492 (a) (1) – (5): Neither international preliminary examination fee (37 CFR 1.482) Nor international search fee (37 CFR 1.445(a)(2)) paid to USPTO And International Search Report not prepared by the EPO or JPO .....**$970.00** International preliminary examination fee (37 CFR 1.482) not paid to USPTO but International Search Report prepared by the EPO or JPO .....**$840.00** International preliminary examination fee (37 CFR 1.482) not paid to USPTO but international search fee (37 CFR 1.445(a)(2)) paid to USPTO .....**$690.00** International preliminary examination fee (37 CFR 1.482) paid to USPTO But all claims did not satisfy provisions of PCT Article 33(1) – (4).....**$670.00** International preliminary examination fee (37 CFR 1.482) paid to USPTO and all claims satisfied provisions of PCT Article 33(1) – (4).....**$96.00**		
ENTER APPROPRIATE BASIC FEE AMOUNT =	$ 690.00	
Surcharge of **$130.00** for furnishing the oath or declaration later than ☐ 20 ☐ 30 Months from the earliest claimed priority date (37 CFR 1.492(e)).	$ -0-	

CLAIMS	NUMBER FILED	NUMBER EXTRA	RATE		
Total claims	30 -20 =	10	X **$18.00**	$180.00	
Independent claims	2 -3 =	0	X **$78.00**	$ -0-	
MULTIPLE DEPENDENT CLAIM(S) (if applicable)			+ **$260.00**	$ -0-	
TOTAL OF ABOVE CALCULATIONS =				$ 870.00	
Reduction of ½ for filing by small entity, if applicable. A Small Entity Statement must also be filed (Note 37 CFR 1.9, 1.27, 1.28).				$ 435.00	
SUBTOTAL =				$ 435.00	
Processing fee of **$130.00** for furnishing the English translation later than ☐ 20 ☐ 30 Months from the earliest claimed priority date (37 CFR 1.492(f)). +				$ -0-	
TOTAL NATIONAL FEE =				$ 435.00	
Fee for recording the enclosed assignment (37 CFR 1.21(h)). The assignment must be Accompanied by an appropriate cover sheet (37 CFR 3.28, 3.31). **$40.00** per property +				$ -0-	
TOTAL FEES ENCLOSED =				$ 435.00	
				Amount to be refunded:	$
				Charged:	$

a. ☒ A check in the amount of $ 435.00 to cover the above fees is enclosed.

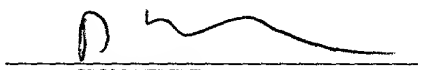
b. ☐ Please charge my Deposit Account No. _____ in the amount of $_____ to cover the above fees. A duplicate copy of this sheet is enclosed.

c. ☐ The Commissioner is hereby authorized to charge any additional fees which may be required, or credit any overpayment to Deposit Account No. _____. A duplicate copy of this sheet is enclosed.

NOTE: Where an appropriate time limit under 37 CFR 1.494 or 1.495 has not been met, a petition to revive (37 CFR 1.137 (a) or (b) must be filed and granted to restore the application to pending status.

SEND ALL CORRESPONDENCE TO:

DR. D. GRAESER LTD.
C/O THE POLKINGHORNS
9003 FLORIN WAY
UPPER MARLBORO
MARYLAND 20772
USA



SIGNATURE

D'VORAH GRAESER
NAME

40,000
REGISTRATION NUMBER

PTO/PCT Rec'd 05 FEB 2002
09/937315

IN THE US PATENT AND TRADEMARK OFFICE

Inventor:
HADAS et al.

Application Ser. No.: 09/937,315

Filing Date: September 24, 2001

For: AUTOMATIC INTERFACE GENERATION FOR AN ENTERPRISE RESOURCE

Attorney Docket: I01/3

Commissioner of Patents and Trademarks
Washington, D.C. 20231
USA

09937315 050202

PRELIMINARY AMENDMENT

Sir:

This is a Preliminary Amendment in the above-referenced case. Please amend this application as follows:

please insert the following text below the title (on line 3, page 1): --This application claims priority from PCT Application No. PCT/IL00/00178, filed on March 20, 2000, which claims priority from US Patent Application No. 09/273,464, filed on March 22, 1999.--

Respectfully submitted,

D'vorah Graeser
US Patent Agent
Reg. No. 40,000

Date: January 31 2001

09/937315
JC16 Rec'd PCT/PTO SEP 2 4 2001

AUTOMATIC INTERFACE GENERATION FOR AN ENTERPRISE RESOURCE

FIELD AND BACKGROUND OF THE INVENTION

The present invention relates to a system and method for automatically generating an application for interacting with an enterprise resource, and in particular, for such a system and method which constructs such an application without extensive manual intervention from a human programmer for an enterprise resource, such that the data associated with the enterprise resource is available through the application, for manipulation, export and import of data.

An enterprise resource is a structured set of data which is accessible through an enterprise application, for example by manipulating, exporting and importing data. The enterprise application is a software program which enables such access and manipulation of the data of the resource. One example of an enterprise resource is a database which is accessible through an interface written in the COBOL programming language. The term "enterprise" refers to an existing resource and application within an organization, for example. Such enterprise resources may be maintained simply because constructing a completely new data structure and a new application interface is expensive and time-consuming, and may also result in significant "down time" until the new application is running smoothly and the enterprise application can be removed. Thus, organizations are hesitant to completely replace enterprise applications and resources.

A more useful system and method would enable the organization to integrate new or additional functionality to the enterprise application and resource, while still maintaining the enterprise application and resource, without requiring extensive programming by a human programmer and without requiring extensive testing before the new application and data structure is ready for use. Such a system and method would be either semi-automatic, or even completely automatic, thereby reducing the possibility of human error as well as reducing the amount of human intervention required to prepare the new application and resource. Finally, such a system and method would produce an application which is widely usable across many different computer platforms, yet which is flexible and robust. Unfortunately, such a system and method are not currently available.

There is thus a need for, and it would be useful to have, a system and a method for automatic generation of a new resource data structure and of a software interface for that data structure, which would interface with an enterprise resource and an enterprise application, and

which would be flexible and robust, yet would not require substantial manual intervention by a human programmer for the construction of the new data structure and software interface, thereby integrating the enterprise application or applications into a single client interface.

BRIEF DESCRIPTION OF THE DRAWINGS

The foregoing and other objects, aspects and advantages will be better understood from the following detailed description of a preferred embodiment of the invention with reference to the drawings, wherein:

FIG. 1 is a schematic block diagram of an illustrative system and work flow according to the present invention; and

FIG. 2 is a schematic block diagram of an application created according to the present invention.

SUMMARY OF THE INVENTION

The present invention is of a system and a method for automatic creation of a software application for accessing an enterprise data resource, preferably without actually altering the stored data. Rather, the system and method of the present invention create an interface which is preferably operable across computer platforms, and which enables the user to both write data to, and retrieve data from the enterprise resource. This is accomplished by first analyzing the enterprise resource to decompose it into a plurality of logical data units. Each data unit is then translated into an application component, which is preferably an object with associated methods and data. Each application component is then added to a hierarchical object-oriented structure, which is constructed according to the logical relationships between the units of data. The hierarchical object-oriented structure is optionally edited manually by the user. Finally, the hierarchical object-oriented structure is transformed into the application by an engine which uses a "factory" model, thereby efficiently coding those portions of the interface which are similar or even identical for all enterprise resources. The final application is preferably a group of objects which provide such functions as a user interface and data input and output.

According to the present invention, there is provided a system for automatic construction of an application for an enterprise resource for operation by a user, the system comprising: (a) a meta-data description of the enterprise resource, the meta-data description featuring a plurality of logical units of data; (b) a meta-data parser for parsing the enterprise resource into the plurality of logical units of data according to the meta-data description and for

determining a relationship between the logical units of data; (c) a meta-data translator for translating each logical unit of data into an application component, for associating the application component with at least one interface component for interfacing with the enterprise resource; and (d) a definition extraction factory for creating the application from a plurality of the application components with the at least one interface component, at least according to the relationship between the plurality of logical units of data.

According to another embodiment of the present invention, there is provided a method for automatically constructing an application for an enterprise data resource for a user, the steps of the method being performed by a data processor, the method comprising the steps of: (a) providing a meta-data description of the enterprise resource, the meta-data description including at least one attribute of the enterprise resource; (b) dividing the enterprise resource into a plurality of logical data units according to the meta-data description; (c) translating each of the plurality of logical data units into an application component according to at least one attribute of the meta-data description to form a plurality of application components; (d) determining a relationship between the plurality of logical data units; and (e) constructing the application according to the plurality of application components and the relationship between the plurality of logical data units.

Hereinafter, the term "network" refers to a connection between any two computers which permits the transmission of data. Hereinafter, the term "computer" includes, but is not limited to, personal computers (PC) having an operating system such as DOS, Windows™, OS/2™ or Linux; Macintosh™ computers; computers having any type of Java virtual machine as the operating system; and graphical workstations such as the computers of Sun Microsystems™ and Silicon Graphics™, and other computers having some version of the UNIX operating system such as AIX™ or SOLARIS™ of Sun Microsystems™; or any other known and available operating system, including operating systems such as Windows CE™ for embedded systems, including cellular telephones, handheld computational devices and palmtop computational devices, and any other computational device which can be connected to a network. Hereinafter, the term "Windows™" includes but is not limited to Windows95™, Windows 3.x™ in which "x" is an integer such as "1", Windows NT™, Windows98™, Windows CE™ and any upgraded versions of these operating systems by Microsoft Corp. (USA).

Hereinafter, the term "Web browser" refers to any software program, which can display multimedia data such as text, graphics, or both, from Web pages on World Wide Web sites.

Hereinafter, the term "Web page" refers to any document available for download and display, including, but not limited to, documents written in a mark-up language such as HTML (Hyper-text Mark-up Language), VRML (Virtual Reality Modeling Language), dynamic HTML, XML (Extended Mark-up Language) or related computer languages thereof.

Hereinafter, the term "application user" refers to the individual operating the application which is constructed according to the present invention, while the term "programming user" refers to the human programmer who is constructing the application by interacting with the system of the present invention.

The present invention could be described as a series of steps implemented by a data processor, such that the present invention could be implemented as hardware, software or firmware, or a combination thereof. For the present invention, a software application could be written in substantially suitable programming language, which could easily be selected by one of ordinary skill in the art. The programming language chosen should be compatible with the computer by which the software application is executed. Examples of suitable programming languages include, but are not limited to, C, C++ and Java.

DETAILED DESCRIPTION OF THE INVENTION

The present invention is of a system and a method for automatic creation of a software application for accessing an enterprise data resource, preferably without actually altering the stored data. Rather, the system and method of the present invention create an application which is preferably operable across computer platforms, and which enables the application user to both write data to, and retrieve data from the enterprise resource. This is accomplished by first analyzing the enterprise resource to decompose it into a plurality of logical data units. Each data unit is then translated into an application component, which is preferably an object with associated methods and data. Each application component is then added to a hierarchical structure which is preferably object-oriented. This hierarchical object-oriented structure is constructed according to the logical relationships between the units of data. By "object-oriented", it is meant that the hierarchical object-oriented structure features a plurality of objects, with data and methods, which are linked according to the relationships between these objects. The hierarchical object-oriented structure is optionally edited manually by the programming user. Finally, the hierarchical object-oriented structure is transformed into the application by an engine which uses a "factory" model, thereby efficiently coding those portions of the interface which are similar or even identical for all enterprise resources. The

final application is preferably constructed as a group of component objects which provide such functions as a user interface and data input and output.

The final application is preferably adjusted to the requirements of a requesting client at run-time, particularly with regard to the GUI. For example, if the client which is requesting access to the enterprise resource data is a thin client operating a Web browser through HTML, then preferably the GUI is implemented as a Web page with HTML forms. More preferably, the GUI is first constructed as a GUI abstraction, thereby enabling different specific, concrete GUI implementations to be constructed "on the fly" as described in greater detail below.

The term "enterprise resource" includes such resources as a database or other data storage structure written in the programming language COBOL, for example, or a terminal emulator interface which enables the application user to access data through a particular type of terminal. Other examples of such enterprise resources are also possible within the scope of the present invention. However, for the purposes of discussion only, and without intending to be limiting in any way, the following description centers upon two types of enterprise resources: a data storage structure written in COBOL, and a terminal emulation software interface.

The principles and operation of a method and system according to the present invention may be better understood with reference to the drawings and the accompanying description, it being understood that these drawings are given for illustrative purposes only and are not meant to be limiting.

Referring now to the drawings, Figure 1 is a schematic block diagram of the workflow through the software modules in a system **10** for automatic generation of a new software application for an enterprise resource, preferably as a hierarchical object-oriented structure. A system **10** initially receives a meta-data description **12** of the data structure of the enterprise resource. This meta-data description divides the data of the enterprise resource into a plurality of logical data units **14**. For example, for a data structure written in the COBOL programming language, each logical data unit **14** would be determined according to the syntax of the COBOL programming language. This syntax divides the data into different types, each of which is stored in a particular field. Thus, for a data structure written in COBOL, one example of a logical data unit **14** would be a particular type of data field.

As another example, for a terminal emulation software program such as a terminal emulation program for the 3270 terminal type, meta-data description **12** would include information obtained from trapping a plurality of screen displays for that terminal. Again, each screen display could be divided into data entry fields, for example, such that each data entry

field could be an example of logical data unit **14**.

Meta-data description **12** for a particular enterprise resource is then received by a meta-data parser **16**. Preferably, each meta-data description **12** has a separate meta-data parser **16**. For example, a COBOL parser **18** would parse meta-data description **12** for a data structure written in the COBOL programming language, while a 3270 parser **20** would parse meta-data description **12** for a data structure written for a 3270 terminal emulation software interface. Other examples of meta-data parsers **16** are possible and are included within the scope of the present invention.

Each meta-data parser **16** divides meta-data description **12** into the plurality of logical units of data **14**. For example, COBOL parser **18** divides meta-data description **12** for a data structure written in COBOL according to the known syntax of the COBOL programming language. Thus, each meta-data parser **16** must be specifically constructed for each meta-data description **12**.

Each logical unit of data **14** is then passed to a meta-data translator **22**. Meta-data translator **22** translates each logical unit of data **14** into an application component **24** during a translation process according to two sets of rules. The first set of rules is a set of internal translation rules **26**, which is defined separately for each type of meta-data description **12**. For example, set of internal translation rules **26** for a data structure written in COBOL are determined according to the known syntax of the COBOL programming language. These rules enable meta-data translator **22** to retrieve such information from logical unit of data **14** as a name for application component **24**, a data type for application component **24**, a display interface for application component **24**, and so forth. Examples of data types for application component **24** include, but are not limited to, a scalar, a Boolean, an integer and a list. Thus, each set of internal translation rules **26** is preferably specifically constructed for each meta-data parser **16**.

In addition, set of internal translation rules **26** enables meta-data translator **22** to create default values for definitions which cannot be determined according to the data available in logical unit of data **14**. These default values are determined according to at least one assumption made in set of internal translation rules **26**. For example, if a data for a particular variable is stored in the data of the enterprise resource, then this assumption could include a range of permissible values for that variable.

Preferably, at least one assumption would enable a mechanism for displaying the data in the GUI (graphical user interface) to be determined according to set of internal translation rules

26. For example, a group name and at least one variable field could be defined in COBOL code written to define the data structure of a particular enterprise resource. The default assumptions of set of internal translation rules **26** could optionally define the display mechanism for the associated data as a panel, with the group name as the title of the panel, and the value or values for the variable displayed in the panel. As another example, if the group name includes the word "occurs", such that the related data has more than one instance, then the data could optionally be displayed in a plurality of "tab cards" on the GUI.

The GUI is therefore preferably determined as an abstract GUI, composed of a plurality of abstract components, such as a check box for example. Each abstract component is more preferably selected from a group of such components, such that the properties of the abstract component are optionally and more preferably automatically known to processes which occur at run-time. Thus, preferably these rules enable assumptions concerning both valid data and mechanisms for displaying the data in the GUI to be defined during the translation process, such that the GUI is defined as a GUI abstraction with a plurality of abstract GUI components.

Set of internal translation rules **26** enables each application component **24** to be created by an application component factory **27** according to the attributes of a template for application component **24**. Application component factory **27** is therefore able to use the information obtained according to set of internal translation rules **26** by meta-data translator **22** to enter the necessary information for each attribute in the template, thereby creating application component **24**. More preferably, application component factory **27** is able to transform each application component **24** into an object, with associated data and methods for accessing the data. This preferred object-oriented architecture for each application component **24** also enables application component **24** to be placed within a hierarchical object-oriented structure for the new enterprise resource, described in greater detail below.

In addition, preferably the interface components (not shown), which are other associated components of the generated application, described in greater detail below with regard to Figure 2, are also generated by application component factory **27**. These interface components are constructed from an interface, which provides the standard for defining each type of interface component. Application component factory **27** selects the correct interface components for each application component **24**, as well as the correct attributes for each interface component, according to the translation process which was previously described.

The second set of rules for the translation process is a set of external translation rules **28**. These external rules preferably include at least one rule defined by the programming user

of the system of the present invention, which either override one or more rules of set of internal translation rules **26**, or alternatively are in addition to one or more rules of set of internal translation rules **26**. Optionally and preferably set of external translation rules **28** includes at least one rule which is specific to the particular enterprise resource but which is not necessarily defined according to the syntax structure of the enterprise application. For example, such a rule could modify the length of the data or could define particular features of the mechanism for displaying the data on the GUI.

Meta-data parser **12** also determines a position **30** for each logical unit of data **14** within the hierarchical object oriented structure. Meta-data parser **12** preferably builds the tree dynamically, as each logical unit of data **14** as parsed, by determining the relationship between each parsed logical unit of data **14** and previously parsed logical units of data **14**, for example as a parent, brother, or child of such a previously parsed logical unit of data **14**.

Next, logical position **30** for each logical unit of data **14** is used to construct a hierarchical object-oriented structure **32** of application components **24** by a hierarchical structure constructor (not shown), which also includes the associated interface components for each application component **24**. Hierarchical object-oriented structure **32** is preferably automatically adjusted to reduce repetition of data. Optionally and preferably, if a particular application component **24** recurs repeatedly throughout meta-data structure **12**, preferably hierarchical object-oriented structure **32** is constructed to indicate the repeated occurrences of application component **24** without repeating all of the information about application component **24**, for example by storing a pointer to application component **24**.

Hierarchical object-oriented structure **32** is optionally displayed to the programming user by an editing environment (not shown), such that the programming user can then edit hierarchical object-oriented structure **32**. For example, the programming user could choose to alter a position of one or more application components **24**, to remove one or more application components **24** from hierarchical object-oriented structure **32**, to adjust one or more associated interface components, or to add information which could not be automatically determined by meta-data translator **22** or application component factory **27**. One example of such information would be specific values for one or more parameters for which default values were entered during the automatic translation process described previously. Such information is preferably entered by the programming user, since values for these parameter(s) are not available from the data in logical unit of data **14**, or else cannot be determined automatically during the translation process. More preferably, hierarchical object-oriented structure **32** is displayed to the

programming user in the context of a development environment, such as a GUI (graphical user interface), which simplifies these changes and additions through "drag and drop" GUI gadgets and other aids to the programming user.

One advantage of system **10** of the present invention is that the rules of set of internal translation rules **26** is optionally simplified, in order to avoid complex rule-based systems which are difficult to operate. Rather, system **10** both enables the programming user to define additional, enterprise resource-specific rules in set of external translation rules **28**, and to manually edit hierarchical object-oriented structure **32**. Thus, system **10** both is simple to operate and yet still enables the programming user to adjust the constructed application as needed.

Hierarchical object-oriented structure **32** of application components **24** is then preferably transformed by a definition extraction factory **34** into a definitions file **36**. Definition extraction factory **34** travels down through hierarchical object-oriented structure **32** and examines each application component **24** individually. Definition extraction factory **34** then extracts the necessary information from each application component **24** in order to construct definitions for each aspect of the new application for accessing the enterprise resource. Definition extraction factory **34** preferably also selects the correct interface components, with the correct attributes, for each application component **34**. More preferably, definition extraction factory **34** includes a plurality of creators (not shown). Each particular creator is able to extract information from application component **24** in order to select the correct instance of an interface component, with the correct attributes, for a particular type of interface component.

Definitions file **36** is preferably a flat text file containing a list of these definitions for each application component **24** and for the associated interface components. However, definitions file **36** could also be a generic text file or a binary format file, for example. Definitions file **36** is then used to create the internal structure of application **38** at run-time, such that the definitions for each application component **24** are read, and are then used to create application **38**.

Preferably, these definitions are parsed by a parsing engine (not shown) which is similar to application component factory **27**. Since these definitions are determined according to the present invention, there is no requirement for translating the data structure or for assuming any default parameters. Instead, the parsing engine creates each application component **24**, and then links each application component **24** to the interface components as described in greater

detail below. The overall structure of hierarchical object-oriented structure **32** is then created by the parsing engine to link all of these components together into application **38**.

More preferably, parsing engine creates a framework for the components of application **38**. This framework then optionally provides services for these components. Ultimately, the enterprise data is provided to a client (not shown), which then displays the data to the user. Therefore, one portion of the framework is most preferably an object which contains the client information, such as the type of communication language used by the client, as well as the type of GUI requested by the client. Examples of different types of communication languages include, but are not limited to, HTML (HyperText Mark-up Language); Java, which may have different sub-types such as Swing, AWT and so forth, XML (extensible mark-up language); and WML (Wireless Mark-up Language). Examples of different types of GUI include, but are not limited to, a Web browser, a Java client interface and no GUI. The latter type of GUI is preferred for applications, such as SQL (sequential query language) database applications by Oracle Ltd. or other such applications, which do not require a GUI. Thus, the client could optionally request not to receive a GUI for these types of applications, but otherwise preferably specifies a type of GUI for displaying the information.

Once the parsing engine receives this information, then the GUI is generated at run-time and/or the time of parsing (if this process occurs before the run-time for the client). Preferably, the GUI is generated by the parsing engine from the previously described GUI abstraction, with the abstract GUI components. The parsing engine also preferably requests a particular library of corresponding specific, implemented GUI components according to the type of communication protocol and the type of application. For example, one type of library might be relevant to a GUI designed for a Web browser and implemented in HTML, while a different type of library would be used for a Java interface GUI which is implemented in the Swing type of Java. Each library would contain information for implementing each type of abstract GUI component as an actual, concrete GUI component.

In addition, each library preferably includes a layout manager for determining the spatial relationship between GUI components. More preferably, each such component, whether concrete or abstract, is specified in relation to other GUI components, rather than with absolute sizes or other absolute properties, in order for the layout manager to operate more effectively. The layout manager is also optionally and more preferably embodied in a rules-based implementation, for determining properties of the GUI component such as size for example. Such a layout manager is available as part of the Java developers' toolkit (Sun Microsystems

Ltd.), but is also preferably implemented for other communication languages such as HTML for example.

Optionally and more preferably, the layout manager operates according to a selected layout policy, which again is most preferably abstracted to an abstract policy, similar to the GUI abstraction. Such an abstract policy enables the parsing engine, in combination with the layout manager, to more easily implement the actual GUI with the actual GUI layout policy, regardless of the client communication language type and/or the client application type. One advantage of such a set of abstractions is that the human software developer can more easily construct various types of GUI components and layout policies as a generalized GUI, rather than implementing the GUI specifically for each type of client communication language type and/or the client application type.

More preferably, at run-time adjustments are made to the structure of application **38** by the parsing engine. For example, the parsing engine may recognize a repetitive sequence within hierarchical object-oriented structure **32**, with multiple sub-hierarchies associated with a particular application component **24**. The parsing engine could then optionally reproduce portions of application **38** which correspond to the multiple sub-hierarchies, thereby more efficiently generating application **38**. Preferably, the programming user can also optionally use this feature of system **10** in order to create part of hierarchical object-oriented structure **32** by selecting and then duplicating one or more application components **24**.

Each interface component of application **38**, apart from application component **24**, is preferably defined according to an interface for that type of component. Preferably, the programming user is able to select interface components from a predefined set of interface components in order to construct application **38**. Optionally and preferably, the programming user can create new interface components from the interface which is provided for each type of component, such that these new interface components are specific to the requirements of the particular enterprise resource for which the programming user is constructing application **38**.

If the programming user wishes to alter one or more aspects of application **38** after definitions file **36** has been created, then definitions file **36** is read by definition extraction factory **34** to form all of the components, including application component **24** and the interface components of application **38** (not shown in Figure 1, see Figure 2), as objects. Each object can then be adjusted by the programming user, for example by altering one or more rules of set of external translation rules **28**. The object is then processed again by application component factory **27** to form application component **24** and/or one or more of the interface components of

application **38**.

Figure 2 shows a preferred implementation of the structure of application **38**, which features components which can be divided into three levels of control of functions within application **38**. In addition, the components are of two types. The first type of component is the interface components which were previously described. The second type of component provides the backbone for application **38**, and includes application component **24** and extensions to application component **24**.

The first of the three levels is a base level **40**, which provides overall control for application **38**. The backbone component of base level **40** is a base application component **42**, which manages the interactions of the remaining components of application **38**. Base application component **42** is the most extended from application component **24**, and is preferably unique to application **38**. Base application component **42** controls both the flow of data, including data import to and export from the enterprise data resource, and the display of the user interface to the application user.

An application control component **44** contains all of the rules and activities for importing data to, and exporting data from, the enterprise resource, which are preferably at least partially determined by the programming user. Preferably, the programming user enters a set of rules for how the enterprise resource can be accessed, for example for displaying certain types of data, and application control component **44** is then created at least partially according to these rules.

A user interface control component **46** provides control for the user interface, including for the functionality of the user interface and for the way in which the user interface is displayed to the user. Both application control component **44** and user interface control component **46** are controlled by base application component **42**.

The next level of application **38** is a first node level **48**. First node level **48** contains components for controlling data import to, and export from, the enterprise resource and for display of the retrieved data to the application user through the user interface. The backbone component for first node level **48** is an I/O component **50**, which is also an extension of application component **24**, although somewhat less extended in comparison to base application component **42**. I/O component **50** controls the activities of an I/O control component **52** and of a user interface component **54**, as well as passing data to be exported from the enterprise resource and displayed by the user interface according to user interface component **54**, and data to be retrieved through the user interface and imported to the enterprise resource through I/O

control component **52**.

I/O control component **52** controls data input and output from the enterprise resource. Optionally and preferably, each application **38** has more than one I/O control component **52** for retrieving data from the enterprise resource according to different retrieval modes. For example, the data could be presented to the application user through a terminal emulation software interface or alternatively through a data structure interface written in COBOL.

Preferably, application **38** is coded as a group of Java-based data objects, since the Java programming language enables each I/O control component **52** to subscribe to particular types of events, and hence to "listen" only for activity which should be controlled by that particular I/O control component **52**.

User interface component **54** preferably includes instructions for displaying the retrieved data in a GUI container according to the attributes provided by I/O component **50**, such as whether the data is displayed in a table, through a set of "tab card" GUI display objects, or any other type of GUI display form. More preferably, the programming user is able to select a particular pattern for the display of the information in a GUI display object, such that user interface component **54** then displays the data according to the selected pattern. For example, if the data includes a number of repeated sequences of numbers of other data, preferably a pattern is selected which is suitable for repetitive data. Alternatively and preferably, such a pattern could be automatically selected during the previously described automatic conversion process. In addition, user interface component **54** preferably also includes at least one display property, such as color of text or symbols, size of GUI display object, background color, and other display properties for the data.

User interface component **54** optionally and preferably implements the user interface as a Java application, or alternatively as an applet. More preferably, user interface component **54** implements the user interface in a document mark-up language, such as HTML or XML for example, for display by a Web browser.

The next level of both definitions file **36** and application **38** is a second node level **56**. Second node level **56** contains objects which are specific to each application component **24**, including application component **24** itself. In addition, second node level **56** includes an import interface **58**, for importing data into the enterprise resource associated with application component **24**, and an export interface **60**, for retrieving data from the enterprise resource associated with application component **24**. Both import interface **58** and export interface **60** interact with I/O control component **52** of first node level **48**, which was previously described.

Optionally and preferably, a single application component **24** can have more than one associated import interface **58** and export interface **60**, for example in order to enable the enterprise resource to be accessed through multiple types of data access. However, preferably, each associated import interface **58** and export interface **60** is controlled by a single I/O control component **52**.

For the sake of symmetry, user interface component **54** is shown again in second node level **56**. It should be noted that user interface component **54** is present at both first node level **48** and second node level **56**, since the user interface is preferably provided through a single interface component as shown. However, optionally user interface component **54** could be divided into two interface components: a first such component for first node level **48**, which would provide a user interface for controlling data input and output; and a second such component for second node level **56**, which would actually display the exported data.

The operation of the components of first node level **48** and second node level **56** is preferably implemented as follows. I/O control component **52** controls the flow of data into and out from the enterprise resource, notifying the other components of application **38** which are subscribed to events related to particular types of data. For exporting data from the enterprise resource, I/O control component **52** receives the raw data from the resource. I/O control component **52** then analyzes the raw data, and notifies each export interface **60** which is subscribed to a portion of the raw data that such data is available. I/O control component **52** parses the data according to the data structure. For example, for data stored in a structure written in the COBOL programming language, the parsing instruction is based upon a fixed format which includes an offset and a length of the data field. Alternatively, the data may be separated by data separators, such as commas. Also alternatively, the data may be stored in the format "field name = field value". After parsing the data, I/O control component **52** then provides the appropriate portion of the raw data to each export interface **60**, which translates the raw data into the appropriate format for display through user interface component **54**.

Similarly, for importing data into the enterprise resource, user interface control component **46** informs the relevant I/O control component **52** to prepare to receive data for storage. I/O control component **52** then notifies all other subscribing components to provide data through export interface **60**. I/O control component **52** receives the data and prepares the data in the proper format for storage. Optionally and preferably, each I/O control component **52** sends such data to a different computer, through a different interface and based upon a different language. Thus, each I/O control component **52** could be used to control data input to,

and output from, a different enterprise resource, such that application **38** would provide an integrated solution to the management of these different enterprise resources.

A validation interface component **64** interacts with export interface **60**, import interface **58**, user interface component **54** and with a data structure interface component **66**, in order to validate the data associated with application component **24**.

Data validation includes such functions as determining whether data to be retrieved from, or written to, a enterprise resource associated with application component **24** has a value or values which are valid for that type of data. In addition, data validation may include determining whether data can be modified, which in turn is optionally determined by the state of application **38**. For example, in one state, application **38** could be determined to be "read only", such that data can only be retrieved from the enterprise resource, but may not be written to the enterprise resource. In this state, import interface **58** should not be permitted to import new data and/or to change existing data. Each state preferably has a set of rules, for determining if data can be displayed, whether a particular object of application **32** is operative, and so forth. Validation interface component **64** and data structure interface component **66** together provide low-level controls for maintaining the rules for each state of application **32**.

For example, export interface **60** could retrieve data from the enterprise resource, which would be stored in data structure interface component **66**. User interface component **54** would then retrieve the data stored in data structure interface component **66** for display to the application user through the user interface. Thus, the mechanism for displaying the data is independent from the data itself, since export interface **60** translates the data into the correct format for display by user interface component **54** and since user interface component **54** does not need to interact directly with export interface **60**.

Furthermore, one or more components of application **38** could be replaced with another component of the same type, without altering the function of the remaining components. Thus, the structure of application **38** which is displayed in Figure 2 provides maximum flexibility for interacting with the enterprise resource.

It will be appreciated that the above descriptions are intended only to serve as examples, and that many other embodiments are possible within the spirit and the scope of the present invention.

PCT/IL00/00178
IPEA/US 26 MAR 2001

WHAT IS CLAIMED IS:

1. A system for automatic construction of a software program to write data to, and to retrieve data from, an enterprise resource for interaction with a user, the system comprising;

(a) a meta-data description of the enterprise resource, said meta-data description featuring at least one attribute of the data of the enterprise resource, said at least one attribute including a structure of the data;

(b) a meta-data parser for parsing the enterprise resource into said plurality of logical units of data according to said at least one attribute of said meta-data description, and for determining a hierarchical structure for said logical units of data according to said structure of the data;

(c) a meta-data translator for translating each logical unit of data into an application component, such that the data of said logical unit of data is associated with said application component, and for associating said application component with at least one interface component for interfacing with the enterprise resource; and

(d) a definition extraction factory for creating the software program from a plurality of said application components with said at least one interface component, at least according to said hierarchical structure for said plurality of logical units of data.

2. The system of claim 1, wherein the software program features a plurality of interface components, including at least one interface component for data flow to and from the enterprise resource and at least one interface component for providing a user interface to the user.

3. The system of claim 2, wherein the software program further comprises:

(i) a first layer of interface components for providing control of the software program, including at least one interface component for controlling said user interface and at least one interface component for controlling said data flow;

PCT/IL00/00178
IPEA/US 26 MAR 2001

(ii) a second layer of interface components for providing said data input and said data output, including said at least one interface component for providing said user interface; and

(iii) a third layer of interface components for specifically interacting with the enterprise resource.

4. The system of claim 3, wherein said application component is contained in said third layer and said third layer further comprises:

(1) an export interface for retrieving data from the enterprise resource; and

(2) an import interface for storing data into the enterprise resource.

5. The system of claim 4, wherein each of said first layer and said second layer includes an extension of said application component for communicating with said at least one interface component related to said user interface and said at least one interface component related to said data flow.

6. The system of claim 5, wherein said at least one interface component is defined according to a standard interface and according to at least one user-defined function.

7. The system of claim 1, wherein said meta-data description, said meta-data parser and said meta-data translator are specific for a particular type of enterprise resource.

8. The system of claim 7, wherein said meta-data translator translates each logical unit of data according to at least one internal translation rule.

9. The system of claim 8, wherein said at least one internal translation rule is defined according to an attribute of said particular type of enterprise resource.

10. The system of claim 9, wherein said at least one internal translation rule further includes at least one default value for a parameter not described in said meta-data description.

IPEA/US 26 MAR 2001

11. The system of claim 10, wherein said meta-data translator also translates each logical unit of data according to at least one external translation rule defined by the user.

12. The system of claim 1, further comprising:

(e) an application component factory for transforming each application component into an object having said at least one method and being associated with the data of said associated logical data unit; and

(f) a hierarchical structure constructor for arranging said plurality of application components into a hierarchical object-oriented structure according to said hierarchical structure for said logical units of data, such that said definition extraction factory also creates the software program according to said hierarchical object-oriented structure.

13. The system of claim 12, further comprising:

(g) a definitions file for defining each of said plurality of application components, such that the software program is stored as said definitions file, said definitions file being constructed by said definition extraction factory.

14. The system of claim 13, further comprising:

(h) an editing environment for editing said hierarchical object-oriented structure by the user, such that the user manually alters at least a portion of said hierarchical object oriented structure.

15. The system of claim 14, further comprising:

(i) a parsing engine for reading said definitions file and for operating the software program according to said definitions file.

16. The system of claim 15, further comprising:

(j) a user interface of the software program for interacting with the user; and

PCT/IL00/00178
IPEA/US 26 MAR 2001

(k) a client computer for operating said user interface, such that said parsing engine constructs said user interface automatically according to at least one property of said client computer.

17. The system of claim 16, wherein said user interface is a GUI (graphical user interface), said GUI being constructed by said meta-data translator with at least one abstract GUI component, the system further comprising:

(l) at least one GUI library for containing a conversion of said at least one abstract GUI component to a concrete GUI component, such that said parsing engine also constructs said user interface automatically according to said GUI library.

18. The system of claim 17, wherein said at least one GUI library contains a plurality of concrete GUI components and a layout manager for determining a spatial relationship between said plurality of concrete GUI components.

19. A method for automatically constructing a software program to write data to, and retrieve data from, an enterprise data resource for a user, the steps of the method being performed by a data processor, the method comprising the steps of:

(a) providing a meta-data description of the enterprise resource, said meta-data description including at least one attribute of the data of the enterprise resource;

(b) dividing the enterprise resource into a plurality of logical data units according to said meta-data description;

(c) translating each of said plurality of logical data units into an application component according to at least one attribute of said meta-data description to form a plurality of application components;

(d) determining a hierarchical structure for said plurality of logical data units; and

(e) constructing the software program from said plurality of application components according to said hierarchical structure for said plurality of logical data units.

20. The method of claim 19, wherein said at least one attribute of said

PCT/IL00/00178
IPEA/US 26 MAR 2001

meta-data description includes a structure of the data in the enterprise resource, such that step (d) is performed according to said structure of the data in the enterprise resource.

21. The method of claim 20, wherein step (c) further comprises the step of:

(i) defining at least one method for each application component, for operating on the data of each logical data unit associated with each application component, such that each application component is an object having said at least one method and being associated with the data of said associated logical data unit.

22. The method of claim 21, wherein step (d) further comprises the step of:

(i) constructing a hierarchical object-oriented structure for said plurality of application components according to said hierarchical structure for said plurality of logical data units.

23. The method of claim 22, wherein step (d) further comprises the steps of:

(ii) providing an editing environment for displaying said hierarchical object-oriented structure to the user; and

(iii) altering at least a portion of said hierarchical object-oriented structure by the user.

24. The method of claim 23, wherein step (c) is performed according to at least one internal translation rule, said at least one internal translation rule being defined according to a type of the enterprise resource.

25. The method of claim 24, wherein step (c) is additionally performed according to at least one external translation rule defined by the user and wherein step (i) of step (c) is performed by translating each of said plurality of logical data units into an application component according to said at least one external translation rule

26. The method of claim 25, further comprising the steps of:

PCT/IL00/00178
IPEA/US 26 MAR 2001

(f) extracting a plurality of definitions for describing each of said plurality of application components and said hierarchical structure; and

(g) storing the software program by storing said plurality of definitions.

27. The method of claim 26, further comprising the step of:

(h) operating the software program according to said plurality of definitions.

28. The method of claim 27, further comprising the step of:

(i) altering the software program by changing at least one of said plurality of definitions.

29. The method of claim 24, wherein the software program is operated by a client computer, and step (e) includes the steps of:

(1) automatically constructing a user interface according to at least one property of said client computer; and

(2) displaying said user interface to the user for interacting with the user.

30. The method of claim 29, wherein step (1) further includes the steps of:

(A) constructing an abstract user interface for displaying data from the enterprise resource and for interacting with the user; and

(B) constructing a concrete user interface according to said abstract user interface and according to said at least one property of said client computer.

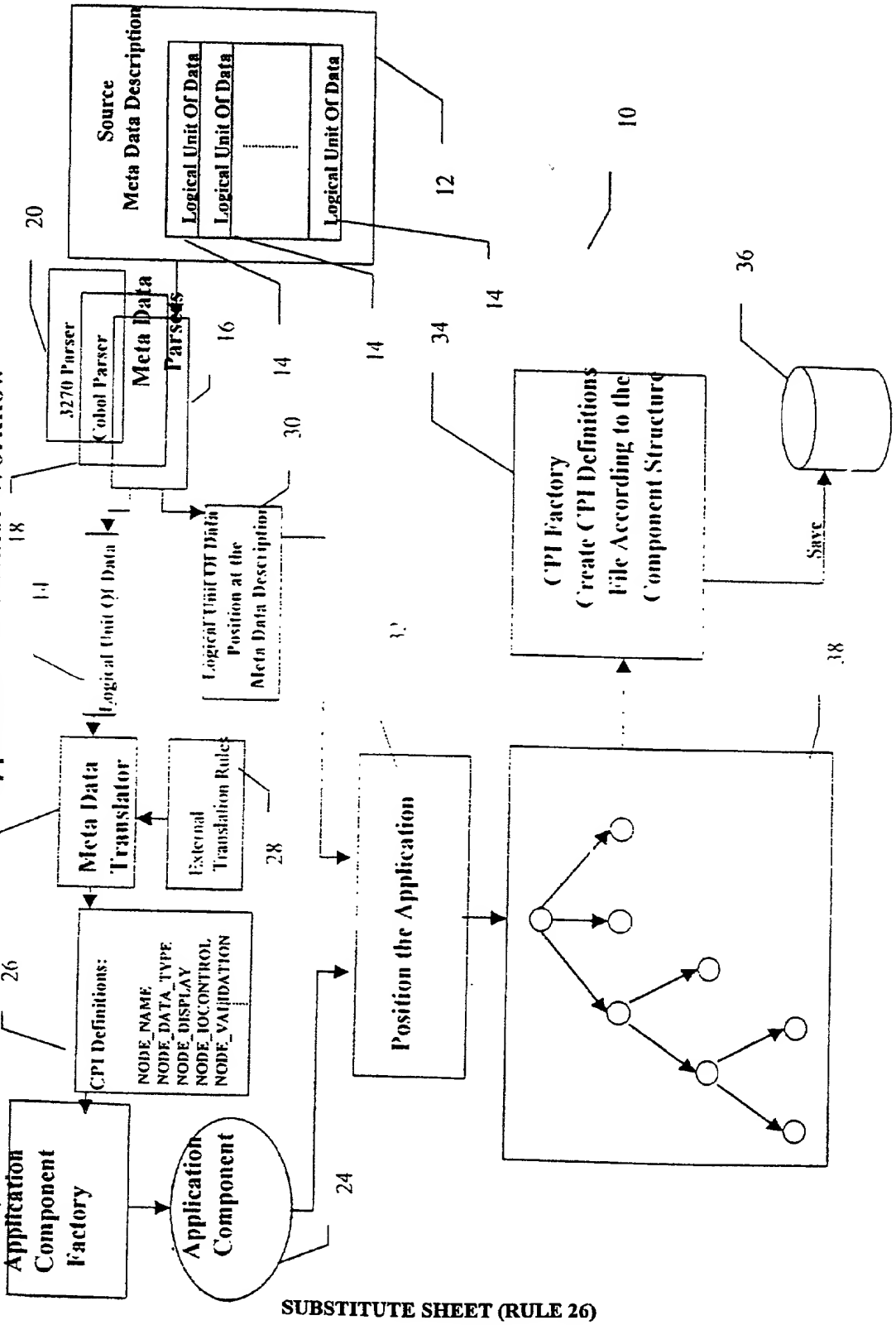


Figure 1
Application Translator Workflow

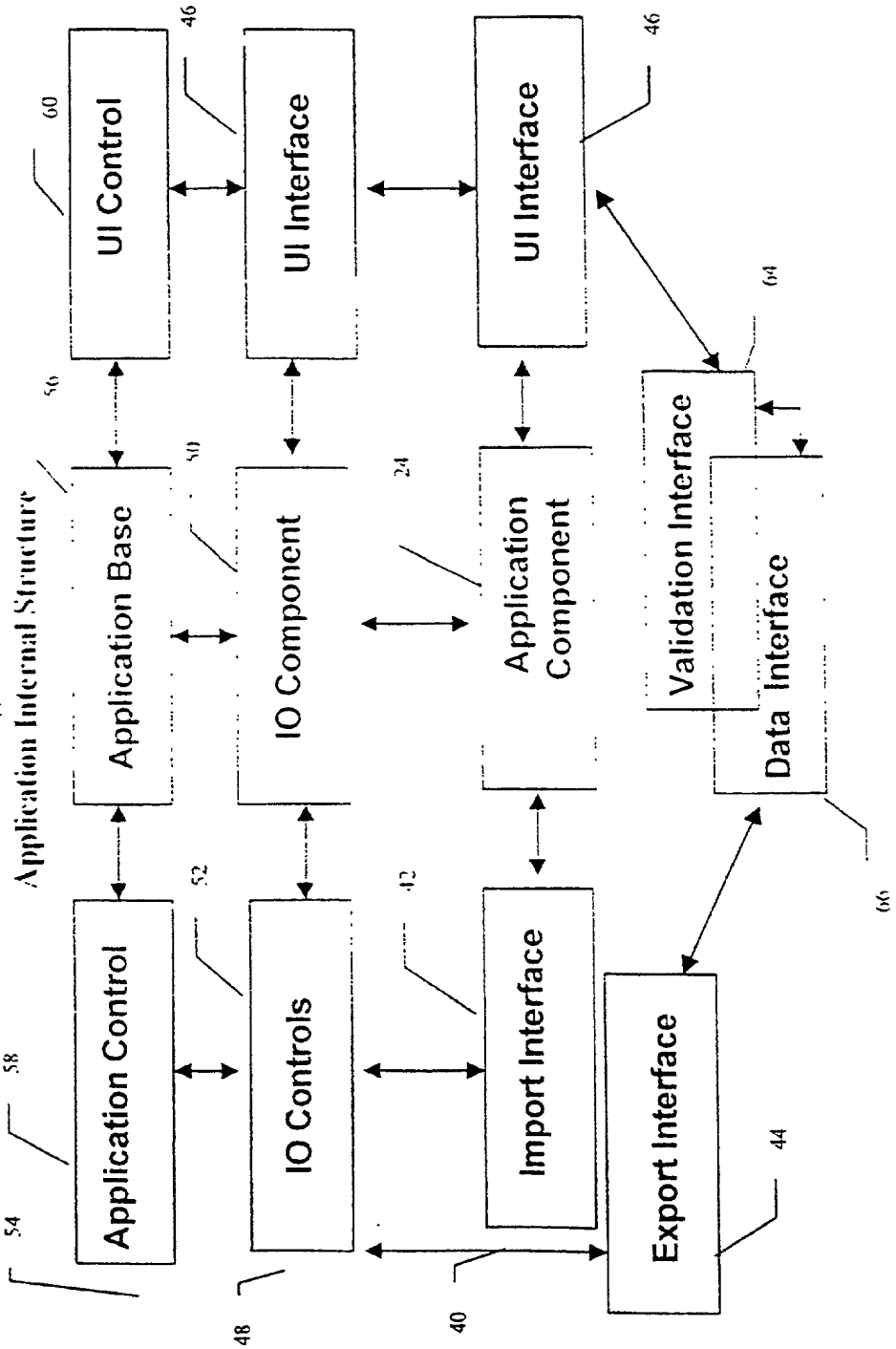


Figure 2
Application Internal Structure

Docket No.
I01/3

Declaration and Power of Attorney For Patent Application

English Language Declaration

As a below named inventor, I hereby declare that:

My residence, post office address and citizenship are as stated below next to my name,

I believe I am the original, first and sole inventor (if only one name is listed below) or an original, first and joint inventor (if plural names are listed below) of the subject matter which is claimed and for which a patent is sought on the invention entitled
AUTOMATIC INTERFACE GENERATION FOR AN ENTERPRISE RESOURCE

the specification of which

(check one)

☐ is attached hereto.

☒ was filed on March 20, 2000 as United States Application No. or PCT International Application Number PCT/IL00/00178

and was amended on _____
(if applicable)

I hereby state that I have reviewed and understand the contents of the above identified specification, including the claims, as amended by any amendment referred to above.

I acknowledge the duty to disclose to the United States Patent and Trademark Office all information known to me to be material to patentability as defined in Title 37, Code of Federal Regulations, Section 1.56.

I hereby claim foreign priority benefits under Title 35, United States Code, Section 119(a)-(d) or Section 365(b) of any foreign application(s) for patent or inventor's certificate, or Section 365(a) of any PCT international application which designated at least one country other than the United States, listed below and have also identified below, by checking the box, any foreign application for patent or inventor's certificate or PCT International application having a filing date before that of the application on which priority is claimed.

Prior Foreign Application(s)			Priority Not Claimed
NA			☐
(Number)	(Country)	(Day/Month/Year Filed)	
(Number)	(Country)	(Day/Month/Year Filed)	☐
(Number)	(Country)	(Day/Month/Year Filed)	☐

I hereby claim the benefit under 35 U.S.C. Section 119(e) of any United States provisional application(s) listed below:

NA	
(Application Serial No.)	(Filing Date)
(Application Serial No.)	(Filing Date)
(Application Serial No.)	(Filing Date)

I hereby claim the benefit under 35 U. S. C. Section 120 of any United States application(s), or Section 365(c) of any PCT International application designating the United States, listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in the prior United States or PCT International application in the manner provided by the first paragraph of 35 U.S.C. Section 112, I acknowledge the duty to disclose to the United States Patent and Trademark Office all information known to me to be material to patentability as defined in Title 37, C. F. R., Section 1.56 which became available between the filing date of the prior application and the national or PCT International filing date of this application:

09/273,464	March 22, 1999	pending
(Application Serial No.)	(Filing Date)	(Status) (patented, pending, abandoned)
PCT/IL00/00178	March 20, 2000	pending
(Application Serial No.)	(Filing Date)	(Status) (patented, pending, abandoned)
(Application Serial No.)	(Filing Date)	(Status) (patented, pending, abandoned)

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

POWER OF ATTORNEY: As a named inventor, I hereby appoint the following attorney(s) and/or agent(s) to prosecute this application and transact all business in the Patent and Trademark Office connected therewith. *(list name and registration number)*

1. **D'VORAH GRAESER** 40,000

Send Correspondence to: **DR. D. GRAESER LTD.**
C/O THE POLKINGHORNS
9003 FLORIN WAY
UPPER MARLBORO, MD 20772, USA

Direct Telephone Calls to: *(name and telephone number)*
THE POLKINGHORNS 301-952-1011

100

Full name of sole or first inventor **ILAN HADAS**
Sole or first inventor's signature Ilan Hadas Date 30/12/01
Residence **HAHADARIM 127, TZORAN 42823, ISRAEL** JLX
Citizenship **ISRAELI**
Post Office Address **HAHADARIM 127, TZORAN 42823, ISRAEL**

Full name of second inventor, if any	
Second inventor's signature	Date
Residence	
Citizenship	
Post Office Address	